REMARKS

Claims 1-26 were submitted for examination in connection with the above-referenced application ("the Application"). All claims stand rejected under 35 U.S.C. 102(b) as being anticipated by "Parallelization of Loops with Exits on Pipelined Architectures", P. Tirumalai et al, Hewlett-Packard Labs, 1990 ("Tirumalai"). In view of the following remarks, Applicant submits that all claims remaining in the case are in condition for allowance and that all other objections have been overcome.

The Office Action fails to present a prima facie case of anticipation for Applicants' claims. "[F]or anticipation under 35 U.S.C. 102, the reference must teach *every aspect* of the claimed invention …" MPEP 706.02 (emphasis added). "The identical invention must be shown in as complete detail as contained in the … claim." *Richardson v., Suzuki Motor Co.*, 868 F. 2d 1226, 1236, 9 USPQ2d 1913, 1920 (Fed. Cir. 1989). Tirumalai simply fails to disclose every aspect of the claimed invention.

Tirumalai is an article that discusses overlapping loops on pipelined computers that issue multiple operations per cycle. Tirumalai discloses scheduling techniques for overlapping "Do" loops at section III, while it discloses scheduling techniques for repeat-until, while and loops with multiple exits at section IV. Tirumalai discloses the following difference between the two types of loops: for the latter category of loops, the trip count is unknown. *See* Tirumalai, p. 205, Section IV, 2nd paragraph ("the loop count is unknown") and p. 200, col. 2, 2nd full paragraph ("these are loops with conditional recurrences"). Thus, Section III discusses scheduling concepts for "Do" loops having a known trip count, while Section IV of Tirumalai discusses scheduling concepts for those loops for which the trip count is unknown. Tirumalai thus generally discusses certain principles relevant to software pipelining of loops. However, Tirumalai does not address the issue of "clobbering" live-in values during speculated execution of loop instructions.

Applicants also generally discuss software pipelining of loops, even referring to the work of P.P. Tirumalai. (See Application, p. 2, line 18 – p. 3, line 3). However, Applicants specifically indicate that a speculatively executed instruction within a software-pipelined loop may modify ("clobber") values that are provided as input to the loop ("live-in values") before the input values are used. Application, p. 3, lines 21-23. Applicants further disclose embodiments of a method for which a speculated instruction may be guarded by a "sticky predicate" that may be set to true once the software pipeline reaches a point at which the speculated instruction will no longer clobber a live-in value to the loop. Application, p. 12, lines 9-14. As the discussion below makes clear, Applicants' claims, as supported by the specification, far exceed the scope of the generalized discussion of software pipelining concepts that are discussed in Tirumalai. All are allowable.

Regarding Claim 1, the Office Action claims that all elements of Claim 1 are disclosed by Tirumalai. More specifically, the Office Action alleges that the italicized portion, below, of Claim 1 is taught at page 205, first paragraph of Tirumalai and also at page 202 of Tirumalai:

1. A method comprising:
initializing to false a predicate that guards a speculative instruction in a software-pipelined loop;

executing at least one iteration of the software-pipelined loop, *including an instruction that sets the predicate to true if an associated live-in value is consumed*; and

executing the speculative instruction in subsequent iterations of the software-pipelined loop.

(See Office Action, p. 3). Such assertion is misguided. As is set forth above, Tirumalai simply does not disclose, suggest or teach *including an instruction that sets the predicate to true if an associated live-in value is consumed.*

The Office Action claims that live-in values are disclosed by Tirumalai at p. 205. Specifically, the Office Action claims that use of "depende ncies" by Tirumalai "i s term for live-in values." Howe ver, such assertion is not supported by the reference itself. The cited portion of Tirumalai indicates that an iteration interval for the overlapping loop may be derived from the data dependence graph (DDG) *for the loop.* Applicants state at p. 3 of the Application that live-in values "are provided as input *t o the loop.*" Application, p. 3, line 23 (emphasis added). While edges of a DDG may correspond to dependencies of a loop (Tirumalai, p. 205, col. 1, first paragraph), a mere reference to a DDG certainly does not disclose, suggest or teach *an instruction that sets the predicate to true if an associated live-in value is consumed* (Claim 1, in part).

The Office Action also states that dependencies are discussed at p. 202, and implies that this discussion allegedly teaches *an instruction that sets the predicate to true if an associated live-in value is consumed* (Claim 1, in part). Such assertion is unfounded.

In the copy of Tirumalai provided to Applicant, the Examiner has pointed out two discussions of dependence (" dependencies" and "depe ndence constraints") on p. 202. The first occurrence states the dependencies for a write operation may be supported through memory, as denoted by the use of "$i" notation. The cited sentence deals with a pseudo-code instruction that writes to memory (See Tirumalai, Fig. 3). Because such statement deals with a write operation, it cannot possibly be read on *consumption* of a live-in value. Obviously, consumption of a live-in value would involve a read operation, not a write operation. Accordingly, the discussion of "dependencie s" on p. 202 does not teach *an instruction that sets the predicate to true if an associated live-in value is consumed* (Claim 1, in part).

The reference to "dependence constraints" o n p. 202 similarly fails to teach the limitation in question. The cited section of Tirumalai merely indicates that overlapping iterations of a loop may be executed "provided resource and dependence constraints are adhered to." (Tir umalai, p. 202, col. 2, 2nd full paragraph). Such statement clearly does not disclose, teach, nor suggest *an instruction that sets the predicate to true if an associated live-in value is consumed*(Claim 1, in part). Claim 1, and all claims depending from Claim 1, are allowable for at least this reason.

Regarding claims 13, 18 and 21, the Office Action wholly fails to even attempt to make a prima facie showing of anticipation. The Office Action merely refers to the rejection of Claim 1. Regarding Claim 13, Applicant is also referred to "repeat of steps 1-4 on page 205." Because this statement is so cryptic, Applicant was unable to evaluate it. As far as Applicants can tell, there are no "steps 1-4" set forth on page 205 of Tirumalai. The four bulleted terminology clarifications at the bottom of column 2 of p. 205 of Tirumalai are not a series of related steps, but are merely a definition of four separate terms: "under", "iteration's predicate", "*wt op*", and " dominates". Accordingly, "repeat of steps 1-4 on page 205" does not present a prima facie case of anticipation for Claim 13 nor for Claim 18 or Claim 21.

In addition, Claims 13, 18 and 21 each include limitations not present in Claim 1. For instance, Claim 13 recites "guarding the speculative i nstruction with a sticky predicate" and "in serting an instruction to set the sticky predicate true at a specified initiation interval of the loop." Such limitations are not recited in Claim 1 and are not addressed in the rejection of Claim 1 at p. 3 of the Office Action. The Office Action thus fails to present a prima facie case of anticipation regarding Claim 13.

Similarly, Claim 18 recites "the specul ative instruction being gated off by a sticky predicate" and "executing an instruct ion that sets the sticky predicate." Such limitations are not recited in Claim 1 and are not addressed in the rejection of Claim 1 at p. 3 of the Office Action. The Office Action thus fails to present a prima facie case of anticipation regarding Claim 18.

Also, Claim 21 recites "the speculati ve instruction being gated off by a sticky predicate" and "execu ting an instruction that sets the sticky predicate." Such limitations are not recited in Claim 1 and are not addressed in the rejection of Claim 1 at p. 3 of the Office Action. The Office Action thus fails to present a prima facie case of anticipation regarding Claim 21.

For the foregoing reasons, Claims 13, 18 and 21 are allowable because the Office Action has failed to make out a prima facie showing of anticipation for such claims. All claims depending from Claim 13, from Claim 18 and from Claim 21 are also allowable for at least this reason.

Regarding Claim 6, the Office Action also asserts that all elements of Claim 6 are disclosed by Tirumalai. More specifically, the Office Action alleges that all limitations of Claim 6 are anticipated at "page 204, disabling and p age 208." (See Office Action, p. 3). Again, such assertion is misguided. Tirumalai simply does not disclose, suggest or teach every limitation of Claim 6.

Claim 6 recites:

6. A method comprising:
initializing a software-pipelined loop to deactivate a speculative instruction;
executing at least one initiation interval (II) of the software-pipelined loop;
activating the speculative instruction; and
executing subsequent IIs of the software-pipelined loop.

Claim 6 does not recite "disabling," so it is not clear to Applicants how the information on page 204 of Tirumalai allegedly anticipates all or part of Claim 6. The text of page 204 describes how a stuff operation may generate, based on the Boolean result of an "if" clause, predicates for operation of the "then" clause. Such text is merely a generalized description of the use of predicates for conditional execution of loop operations. It does not disclose *initializing a software-pipelined loop to deactivate a speculative instruction*, nor does it disclose *activating the speculative instruction.*

Page 208 of Tirumalai similarly fails to disclose, teach or suggest the limitations of Claim 6. The Office Action does not cite a particular portion of page 208. However, Applicants assume that section V, "Compiling and Scheduling Issues" is not relied upon for the rejection, since it addresses compiler and scheduling issues that are tangential to the discussion of overlapping of loops. It is further assumed that the brief paragraph regarding conversion of while loops to repeat-until loops is also not relied upon for the rejection. Also, it is assumed that the section addressing "Loops with multiple-exits" is not relied upon to support the rejection of Claim 6, since its discussion of single and multiple exits points is also tangential to the discussion of overlapping loops. Accordingly, Applications assume for the sake of traversing the rejection of Claim 6 that the rejection relies upon the material above the "While Loops" title in column 1 of page 208. Such section discusses Figs. 6d and 7 as well as Table V of Tirumalai.

Such material addresses execution of a sample overlapped loop. Rather than *teaching initializing a software-pipelined loop to deactivate a speculative instruction*, such figures and table of Tirumalai instead teach the issue of "operations *without* waiting for the correct predicates" (Tirumalai, p. 207, col. 1, first paragraph, emphasis in original). Rather than teaching initializing to *deactivate* a speculative instruction, they teach exactly the opposite – eager execution of an instruction that may not have executed in the source program (See Tirumalai, p. 207 – "[s]uch operations may execute when they wond not have executed in the source program"). Accordingly, Claim 6 is allowable for at least the reasons. In addition, all claims that depend from Claim 6 are also allowable for at least the foregoing reasons.

Regarding Claim 24, the Office Action again takes a shorthand attempt at rejection that fails to meet for requirements for setting forth a prima facie case of anticipation. The Office Action merely refers to the rejection of Claims 1 and 2, along with "page 205 third bullet on right, disable".

The rejection of Claim 2 merely refers to Figure 1. Such reference is puzzling. For instance, Figure 1, which is a block diagram of a machine model, does not teach, suggest, or disclose "*the instruction that sets the predicate true is gated by a stage predicate of the software-pipelined loop.*" (Claim 2, in part). Fig. 1 generally discloses an instruction format for wide instructions having a predicate specifier field, but does not disclose the opcodes or pseudocode for any specific instructions. Nor does Fig. 1 address, teach or suggest a stage predicate. Accordingly, Fig. 1 clearly does not disclose the limitations of Claim 2. Claim 2 is allowable for at least this reason.

Claim 24 does not include the Claim 2 limitation of "*the instruction that sets the predicate true is gated by a stage predicate of the software-pipelined loop.*" (Claim 2, in part). Therefore, it is baffling why the rejection of Claim 2 was invoked in the rejection of Claim 24. In any event, the rejection of Claim 2 fails to make a prima facie case of anticipation for Claim 24.

Claim 24 also includes limitations not recited in Claim 1. For instance, Claim 24 recites "*initialize a software-pipelined loop t o deactivate a speculative instructions*" and "*activa te the speculative instruction.*" Such limitations are not recited in Claim 1 and are not addressed in the rejection of Claim 1 at p. 3 of the Office Action.

The Office Action thus fails to present a prima facie case of anticipation regarding Claim 24, either alone or in combination with the rejection of Claim 2.

Finally, "page 205 third bullet on right, disable" does not read on Claim 1, either alone or in conjunction with the rejections of Claims 1 and 2. The third bullet on the right-hand side of p. 205 of Tirumala is simply a terminology definition for the term "wtop." The "wtop" operation is not recited in Claim 24. As is explained by the Tirumalia references, a wtop operation generates the next iteration's predicate based on a Boolean datum and the current iteration's predicate. The next iteration's predicate may be *not asserted* if the Boolean datum (branch condition) is false or the current iteration's datum is false. (Tirumalai, p. 205). Such iteration predicate pertains to the entire loop iteration, not to a particular instruction. Accordingly, Tirumalai fails to disclose, teach or suggest the limitations of Claim 24. Claim 24, and all claims depending from Claim 24, are allowable for at least the foregoing reasons.

Accordingly, all independent claims are in condition for allowance. For at least the foregoing reasons, all dependent claims are also in condition for allowance.

Applicant respectfully submits that the applicable rejections have been overcome and must all be withdrawn. All claims are therefore in condition for allowance.

Please charge any shortages and credit any overcharges to our Deposit Account No. 02-2666. The Examiner is requested to call Shireen Irani Bacon at (512) 314-0435 it he perceives that a teleconference would advance allowance of this case.

Respectfully submitted,

Dated: June 14, 2004

[illegible]

Gregory D. Caldwell
Registration No. 39,926

12400 Wilshire Boulevard
Seventh Floor
Los Angeles, CA 90025-1026
(408) 720-8300

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail with sufficient postage in an envelope addressed to Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313 on:

6.14.04
Date of Deposit

Derek S. Watson
Name of Person Mailing Correspondence

[illegible] 6.14.04
Signature **Date**